# gdb+gdbserver调试arm-linux程序

一、概述

1、嵌入式Linux的GDB调试环境由Host和Target两部分组成，Host端使用arm - linux - gdb，Target Board端使用gdbserver。
调试时，应用程序在嵌入式目标系统上运行，而gdb调试在Host端

2、远程调试环境由宿主机GDB和目标机调试stub共同构成，两者通过串口或TCP连接。
使用 GDB标准程串行协议协同工作，实现对目标机上的系统内核和上层应用的监控和调试功能。
调试stub是嵌入式系统中的一段代码，作为宿主机GDB和目标机调试程序间的一个媒介而存在。

3、就目前而言，嵌入式Linux系统中，主要有三种远程调试方法，分别适用于不同场合的调试工作：
用ROM Monitor调试目标机程序、
用KGDB调试系统内核、
用gdbserver调试用户空间程序。
这三种调试方法的区别主要在于，目标机远程调试stub 的存在形式的不同，而其设计思路和实现方法则是大致相同的。
而我们最常用的是调试应用程序。就是采用gdb+gdbserver的方式进行调试。
在很多情况下，用户需要对一个应用程序进行反复调试，特别是复杂的程序。
采用GDB方法调试，由于嵌入式系统资源有限性，一般不能直接在目标系统上进行调试，通常采用gdb+gdbserver的方式进行调试。

二、环境

ubuntu12.04
Linaro GCC 4.9-2014.09（gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux）
imx6开发板
网络环境如下：HOST 192.168.1.128  Target: 192.168.1.2
NFS共享目录: mount  -o nolock 192.168.1.128:/home/wang/imx6  /mnt
samba共享目录：wang

三、源码获取交叉编译

1、gdb编译

1)、源码获取解压

http://www.gnu.org/software/gdb/download/
http://ftp.gnu.org/gnu/gdb/
211.95.105.202：3128可以上去的，所有的版本都有啊
http: //ftp.cs.pu.edu.tw/linux/sourceware/gdb/releases/下载
ftp://ftp.gnu.org/gnu/gdb
外网的ftp我经常上不去，国内常见的开源社区的下载频道通常都有下载的
http://download.chinaunix.net/download/0004000/3482.shtml，
我下载的是gdb-7.0.1a.tar.bz2

gdb-7.0.1a.tar.bz2

//但要注意，gdb的版本需要和croostool 相匹配。

拷贝gdb-7.0.1a.tar.bz2 到wang/imx6下

打开终端
cd /home/wang/imx6
tar -xvf gdb-7.0.1a.tar.bz2
mkdir gdb-7.0.1_build
cd gdb-7.0.1_build
mkdir install

2)、编译arm-linux-gdb

gdb-7.0.1使用了autoconf/automake。
因此，通过设置configure脚本的--target，--host,--prefix参数就可以方便的移植到别的平台。
--target指定了需要调试的目标机环境，一般设置为交叉编译器的前缀，
比如--target=arm-linux,
--target=mips-linux，
--target=armv5-linux-uclibc,
--target的缺省值为i386-linux, 也就是i386PC机
--host指定编译后的文件的运行环境，

取值可以是i386-linux或者交叉编译器的前缀，缺省为i386-linux
--prefix指定要安装的目录。

../gdb-7.0.1/configure --target=arm-linux --prefix=/home/wang/imx6/gdb-7.0.1_build/install
检查不到交叉编译工具链。所以换成以下形式
../gdb-7.0.1/configure -target=arm-linux-gnueabihf --prefix=/home/wang/imx6/gdb-7.0.1_build/install
然后，就可以检测到了，估计是因为我的交叉编译工具链的是以arm-linux-gnueabihf这个开头的
make
出现大量错误：xxx set but not used [-Werror=unused-but-set-variable]
解决：
../gdb-7.0.1/configure --target=arm-linux-gnueabihf --prefix=/home/wang/imx6/gdb-7.0.1_build/install --disable-werror
make

3)、安装
make install

2、gdbserver编译、安装
pwd$
/home/wang/imx6/gdb-7.0.1_build/
cd ..
mkdir gdb-7.0.1_gdb_server_build
cd gdb-7.0.1_gdb_server_build
mkdir install
../gdb-7.0.1/gdb/gdbserver/configure --target=arm-linux-gnueabihf --host=arm-linux-gnueabihf --prefix=/home/wang/imx6/gdb-7.0.1_gdb_server_build #--host=arm-linux-gnueabihf交叉编译
出错：
../gdb-7.0.1/gdb/gdbserver/linux-low.c:2916:18: error: storage size of ‘siginfo’ isn’t known
struct siginfo siginfo;
^
../gdb-7.0.1/gdb/gdbserver/linux-low.c:2917:28: error: invalid application of ‘sizeof’ to incomplete type ‘struct siginfo’

cd /home/wang/imx6/gdb-7.0.1_build/
mkdir gdb_server_build
cd gdb_server_build
mkdir gdb_server_install
../../gdb-7.0.1/gdb/gdbserver/configure--target=arm-linux-gnueabihf --host=arm-linux-gnueabihf --prefix=/home/wang/imx6/gdb-7.0.1_build/gdb_server_build/gdb_server_install
出错：
../gdb-7.0.1/gdb/gdbserver/linux-low.c:2916:18: error: storage size of ‘siginfo’ isn’t known
struct siginfo siginfo;
^
../gdb-7.0.1/gdb/gdbserver/linux-low.c:2917:28: error: invalid application of ‘sizeof’ to incomplete type ‘struct siginfo’
解决：
参考http://www.cnblogs.com/zhuyp1015/p/3619589.html
进入到linux-low.c 中，找到相应函数（大概有两个函数），将 struct siginfo 全部换成为 siginfo_t。
暂未结局，估计是struct siginfo没有定义
make ;make install
编译成功
arm-linux-gnueabihf-strip arm-linux-gdbserver
#其中包含了很大一部分的调试信息，我们可以使用arm-linux-strip工具对其进行瘦身工作，使其文件大小变小。

重新下载gdb7.3.1.tar.gz

5、安装arm-linux-gdbserver到开发板
mount -o nolock 192.168.1.128:/home/wang/imx6 /mnt
cp /mnt/gdb-7.0.1_build/gdb_server_build/gdb_server_install/bin/arm-linux-gdbserver /usr/bin/gdbserver

5、测试

1、
在x86上

测试程序helloworld.c

```
#include<stdio.h>
int main(int argc, char **argv)
{
    printf("hello woeld !!\n");

    return 0;
}
```

```
arm-linux-gcc -g HelloWorld.c -o HelloWorld
```

2、在开发板上：

```
pwd$
    /root
```

拷贝helloworld到开发板上

```
cp /mnt/helloworld ./
gdbserver 192.168.1.128:5412 helloworld
```

注意其中的IP地址是x86的IP，

3、在x86上

```
arm-linux-gnueabihf-gdb helloworld
target remote 192.168.1.2:5412
```

注意其中的IP地址是开发板的IP，其中后面的端口号一定要和之前在开发板中设置的端口后一致，这样才能进行通信。

上面的形式说明远程调试开始。

```
(gdb) l
6     *       Revision:  none
7     *       Compiler:  gcc
8     *         Author:  YOUR NAME (),
9     *        Company:
10     * ************************************************************************/
11
12    #include<stdio.h>
13    int main(int argc, char **argv)
14    {
15        printf("hello woeld !!\n");
(gdb) l
16
17        return 0;
18
19    }
20
21
(gdb) b 15
Breakpoint 1 at 0x103da: file helloworld.c, line 15.
(gdb) c
Continuing.

Breakpoint 1, main (argc=1, argv=0x7eb6be54) at helloworld.c:15
15        printf("hello woeld !!\n");
(gdb) c
Continuing.
[Inferior 1 (process 225) exited normally]
(gdb)
```

在开发板上：

```
Remote debugging from host 192.168.1.128
hello woeld !!

Child exited with status 0
GDBserver exiting
```

## 四、gdb+gdbserver几种调试方式

### 1、gdb+gdbserver nfs调试流程

在开发板上

```
mount -o nolock 192.168.1.128:/home/wang/imx6 /mnt
cd /mnt
gdbserver 192.168.1.128:5412 helloworld
    Process helloworld created; pid = 232
    Listening on port 5412
```

在x86上

arm-linux-gnueabihf-gdb helloworld

target remote 192.168.1.2:5412

注意其中的IP地址是开发板的IP，其中后面的端口号一定要和之前在开发板中设置的端口后一致，这样才能进行通信。

上面的形式说明远程调试开始。

```
 (gdb) l
6    *       Revision:  none
7    *       Compiler:  gcc
8    *         Author:  YOUR NAME (),
9    *        Company:
10     * *************************************************************************/
11
12    #include<stdio.h>
13    int main(int argc, char **argv)
14    {
15        printf("hello woeld !!\n");
(gdb) l
16
17        return 0;
18
19    }
20
21
(gdb) b 15
Breakpoint 1 at 0x103da: file helloworld.c, line 15.
(gdb) c
Continuing.

Breakpoint 1, main (argc=1, argv=0x7eb6be54) at helloworld.c:15
15        printf("hello woeld !!\n");
(gdb) c
Continuing.
[Inferior 1 (process 225) exited normally]
(gdb)
```

在开发板上

```
Remote debugging from host 192.168.1.128
hello woeld !!

Child exited with status 0
GDBserver exiting
```

2、gdb+gdbserver+串口调试

在开发板上

```
gdbserver  /dev/ttymxc0 helloworld
```

在x86上

```
arm-linux-gnueabihf-gdb -b 115200  helloworld
target remote /dev/ttyxxx    #/dev/ttyxxx 为开发板与x86所连接的串口在x86下的设备名
```

剩下的操作和nfs方式一样

五、gdb补充

1、常用的GDB命令：

load：装入一个程序

symbol-file：装入符号库文件，可以是用-g参数编译的可执行文件。

f(ile)：指定一个可执行文件进行调试，gdb将读取些文件的调试讯息，如f a.exe

l(ist)：列程序出源文件

r(un) ：装载完要调试的可执行文件后，可以用run命令运行可执行文件

b(reak)：设置断点（break point），如b 25，则在源程序的第25行设置一个断点，当程序执行到第25行时，就会产生中断；也可以使用b funcname，funcname为函数的名称，

当程序运行到断点停下来时，

c(ontinue)：c命令可以另中断的程序继续执行，直到下一个中断点或程序结束

p(rint)：输入某个变量的值，如程序定义了一个int a的就是，p a就会输出aa的当前值
n(ext)：程序执行到断点时中断执行，可以用n指令进行单步执行
s(tep)：程序执行到断点时中断执行，可以用s指令进行单步执行进某一函数
backtrace：输出调用堆栈
q(uit)：退出GDB

使用简化命令,如l,和完整命令list,效果是一样的。

2、需要注意的问题

1）、简化命令f不一定能用，gdb会提示

(gdb) f gprs
No symbol table is loaded.  Use the "file" command.
可用完整命令file代替
(gdb) file gprs
Reading symbols from /home/www.linuxidc.com/arm2410s/gprs...done.

2）、交错调试时我们不使用r(un)来执行程序。

因为执行完target remote命令后，目标板程序已经在运行，所有应该用continue命令而不是run命令。
如果你选择了重新运行程序，gdb会提示不知道怎么运行程序,因为开发板那边gpbserver已经“Killing inferior”退出了.
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/www.linuxidc.com/arm2410s/gprs
Don't know how to run.  Try "help target".

3、多线程调试

1）. 多线程调试可以设置 set follow-fork-mode child/parent

表示在多线程中产生新线程的时候gdb进入到子线程还是父线程。

Gdb多线程调试常用命令：

http://coolshell.cn/articles/3643.html

多线程调试可能是问得最多的。其实，重要就是下面几个命令：

info thread 查看当前进程的线程。
thread <ID> 切换调试的线程为指定ID的线程。
break file.c:100 thread all  在file.c文件第100行处为所有经过这里的线程设置断点。
set scheduler-locking off|on|step，这个是问得最多的。

在使用step或者continue命令调试当前被调试线程的时候，其他线程也是同时执行的，怎么只让被调试程序执行呢？通过这个命令就可以实现这个需求。

off 不锁定任何线程，也就是所有线程都执行，这是默认值。
on 只有当前被调试程序会执行。
step 在单步的时候，除了next过一个函数的情况(熟悉情况的人可能知道，这其实是一个设置断点然后continue的行为)以外，只有当前线程会执行。

参考网址：http://blog.csdn.net/wzw88486969/article/details/18604003